[TAB+21] C. Talcott, S. Ananieva, K. Bae, B. Combemale, R. Heinrich, M. Hills, N. Khakpour, R. Reussner, B. Rumpe, P. Scandurra, H. Vangheluwe:
Composition of Languages, Models, and Analyses.
In: Composing Model-Based Analysis Tools, pp. 45-70. Springer, 2021.
https://www.se-rwth.de/publications

Chapter 4

Composition of Languages, Models and Analyses

Carolyn Talcott, Sofia Ananieva, Kyungmin Bae, Benoit Combemale, Robert Heinrich, Mark Hills, Narges Khakpour, Ralf Reussner, Bernhard Rumpe, Patrizia Scandurra, Hans Vangheluwe

Abstract. This chapter targets a better understanding of the compositionality of analyses, including different forms of compositionality and specific conditions of composition. Analysis involves models, contexts, and properties. These are all expressed in languages with their own semantics. For a successful composition of analyses it is therefore important to compose models as well as the underlying languages. We aim to develop a better understanding of what is needed to answer questions such as "When I want to compose two or more analyses, what do I need to take into account?" We describe the elements impacting analysis compositionality, the relation of these elements to analysis, and how composition of analysis relates to compositionality of these elements.

This core chapter addresses Challenge 1 introduced in Chapter 3 of this book [Hei+21] (*the theoretical foundations* — how to compose the underlying languages, models and analyses).

4.1 Introduction and Problem Statement

To tackle the complexity of systems design and development, it is necessary to use a multitude of models describing certain aspects, or viewpoints, of the system as a whole or of its subsystems. These models may be expressed using formalisms that provide multiple sublanguages, or special purpose formalisms, or both. Understanding the prerequisites for model composition helps to solve challenges in system design. If the models are specified in different languages describing a variety of views, language composition is required. Even if the models are only augmented with variants of extra properties, compositionality of these kinds of properties must be addressed.

Thus language, semantics, and model composition are an important basis to address the question of how to compose analyses. One main question discussed in this chapter is when and how language, semantics or model composition is in accordance with, or orthogonal to, analysis composition.

For analysis of behavioural and/or quantitative aspects of a model of a system or system component, it is important to also provide (a model of) the execution context — information about patterns of use, and about elements that affect the behaviour but are not part of the modelled system. Thus, we need to understand the ways context can be composed with other contexts and with models of the system under study, and how this relates to the properties being analysed (see also Section 2.1.6 of this book [Hei+21]).

This chapter addresses a better understanding of what is needed to answer questions such as “When I want to compose models or analyses, what do I need to take into account?”; “What are the key relations among models of systems, contexts, and properties and their underlying formalisms?” and “What do these relations tell us about composing analyses?”.

The chapter begins with a discussion of core concepts and their interrelations. Section 4.2 and Section 4.3 recall the key aspects concerning the concepts of model and analysis discussed in detail in Chapter 2 of this book [Hei+21]. Section 4.4 takes a broad view of composition and the relations of composition to the elements of analysis, and identifies several forms of composition. Section 4.5 builds on the discussion of core concepts and presents a mathematical framework characterising the relations between models, analyses and results: how analyses compose, and how composition of the models analysed relates to the analysis results. Section 4.6 presents a diverse sample of formalisms, composition issues, and current practice, to illustrate the ideas presented in the earlier sections. Section 4.7 concludes with a summary of the concepts and challenges and suggests promising research directions.

4.2 Brief Overview of Models and their Composition

Chapter 2 of this book [Hei+21] already contains a detailed definition of the basic concepts that are needed to understand this chapter. We therefore just repeat some core concepts here. We refer to the definition by Stachowiak [Sta73] to describe what a *model* is. General purpose languages, such as the *unified modeling language* (UML) [BRJ98], become complex and require analysis techniques to better handle the complexity both of the language and systems described. Unfortunately, also analysis techniques become complex and therefore require to be decomposed.

By definition, a model has a purpose with respect to the original [Sta73], and can play one or several roles with respect to this purpose. An engineering model typically starts by being descriptive, and then, at design time, is viewed as prescriptive.

According to [Com+16], a modelling language defines a set of models that can be used for modelling purposes. Various forms of syntax are possible. The semantics can, for example, be defined in the denotational form [HR04]. As discussed in Chapter 2 of this book [Hei+21], a sound semantic definition is very helpful to understand what shall be analysed and what the desired outcomes of analysis techniques are.

In model-based analysis, interesting properties can vary. Thus, we use property models in an explicit language with their own precise semantics. In the very same spirit, we use context models to describe entities of the context, outside of the system to be built.

Composition of models in various forms is a key to cope with complexity, but not easily achievable. Furthermore, advanced and potentially integrated forms of composite semantics, need composition of models of different aspects, their modelling languages and finally also their associated analysis techniques.

We use metamodelling technology [Gro06] in constructive as well as analytical tools to manage models in an accessible form. [Com+16] defines: “A *metamodel* is a model describing the abstract syntax of a language”. Composing models described in heterogeneous languages requires a composition of the metamodels in a useful way. Assuming that metamodels are class diagrams, we therefore have several alternatives for integration: merging algorithms, mappings between metamodels, and consistency relations between the metamodels. Constructive algorithmic translations as well as relations between models can be defined explicitly using a *transformation model*.

4.3 Brief Overview of Analysis

Analysis is the process of answering questions about a system under study. The system may be too complex to reason directly about it, or it may not yet exist. Thus, analysis techniques work

with models: models of (some aspect of) the system of interest, of its context, and of the question being asked, i.e., a property.

As proposed in Section 2.2 of this book [Hei+21], the idea of analysis can be captured formally by the relation

$$M, C \vdash_T Q \rightsquigarrow A$$

where M, C, Q, and A are (respectively) models of the system, context, question, and answer domains, and T is an analysis technique.

Analysis can be characterised along multiple dimensions. One dimension is the *level of automation*. At one end of the spectrum, determining whether a property holds may be a fully automatic process, while at the other it may involve informal social processes. Many techniques involve user guided automation. Another dimension is whether the analysis is *static* or *dynamic*. A static analysis works over the syntax of the input models, and usually happens at design time. A dynamic analysis occurs during system or model execution, and may be online (monitoring) or offline (analysis of traces from logged information). Simulation sits on the borderline.

The answer domain of an analysis can be simply a two-element set reflecting success or failure. This is referred to as *Qualitative analysis* and includes checking satisfaction of a given property. Alternatively, in a *Quantitative analysis*, the answer domain is richer: real numbers, a probability distribution, or even tables and other structured data are used. Performance analysis is an example of quantitative analysis.

Similarly to a model having a purpose, an analysis also has a purpose. We distinguish three main *kinds* of purpose: analysis of model/system structure; analysis of functional aspects of behaviour; and analysis of quality aspects of behaviour. The analysis of structure works with syntactic descriptions, while the analysis of behaviour requires a semantic domain (and possibly other information). The purpose of a specific analysis may be a mixture of these basic kinds of purpose. Table 2.1 and Table 2.2 of Chapter 2 of this book [Hei+21] summarise the different purposes and the elements (e.g., model, context) required for analysis.

Analysis techniques can be characterised by how helpful they are. When the answer produced by an analysis is different from what is expected/desired (e.g., type inference fails, safety or security property fails, or a performance measure is out of desired bounds), does the analysis technique provide a reason for failure? Does it help to locate the cause? Does it help to correct the problem? See Chapter 7 of this book [Hei+21] for more discussion of tools' outputs and their use.

Finally, an important consideration is the *quality* of an analysis. This includes different notions of soundness: Does the analysis always give an answer? Is the answer an over- or under-approximation? Does it produce false positives or false negatives? These represent trade-offs of complexity and accuracy. Another quality issue is whether the analysis is repeatable (by the same analyst) or reproducible (by an independent analyst).

4.4 What is Composition?

Figure 4.1 shows a holistic vision of composition of analyses across different syntactic and semantic domains and corresponding properties of interest. In particular, disparate models of different aspects are the main subjects to be composed/decomposed on the syntax and semantic level and also at the metamodel level. These models include: system models, context models, property models, and models of analysis results. Composition of analyses relates to compositionality on the syntax and semantic level of the underlying formalisms to represent (sub)system models and contexts (F_i) and property models (PF_i). The act of such compositions (the operation $COMP$) forms a composite model formalism ($COMP(F_1, F_2)$) and a property formalism ($COMP_F(PF_1, PF_2)$). The composition of model formalisms and property formalisms enables *global analysis* [Cla+14]. Given a system model composed with an intended environment of use, one can formulate analysis questions that apply to a certain set of the individual submodels. These analysis questions can

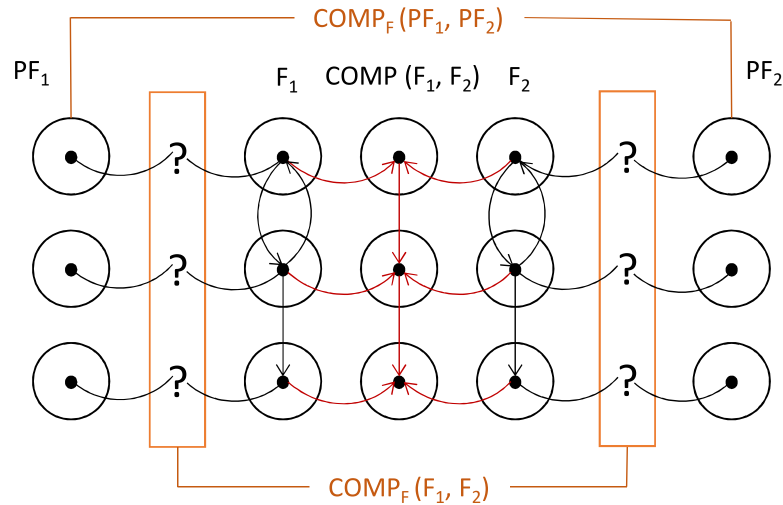

Figure 4.1: Multiple dimensions of composition. The three central columns represent two modelling formalisms, F_1 and F_2 and their composition. Think of the top row as a metamodel, the middle row as a system or component (syntactic) model, and the bottom row as a semantic model. The outer columns represent properties, at each level. The question marks ('?') stand for satisfaction relations. The arrows connecting nodes (dots or question marks) represent relations such as refinement, abstraction, satisfaction, instanceof, meaningof, etc.

be managed at the level of the submodels and contexts involved, by defining and applying appropriate composition/decomposition relational operators (e.g., merge, union, focus, restriction, etc.). These operators are grounded on the semantic domains of the composed formalisms and their supported analysis techniques.

In Section 4.4.1 we give examples of targets of composition, and in Section 4.4.2 we characterise different forms of composition.

4.4.1 Targets of Composition

Based on the concepts described in the previous sections, for the purpose of composing modelling languages and formalisms to enable global analysis, it is necessary to think about the elements that are the targets of composition. These include:

1. Components (of the system under study): architectural, functional, behavioural
2. Models of aspects of the system or its components
3. Application domain — communications, image processing, manufacturing, chemical process control, ...
4. User-facing language composed of several elements from sublanguages
5. Analysis formalisms and techniques (possibly made of several subanalyses formalisms and techniques) such as constraint solving, unification, model checking, or simulation
6. Syntactic domains and semantic domains of all specification languages / formalisms involved
7. Tools composed of several subtools dealing with sublanguages (or subanalyses)

4.4.2 Forms of Composition

Considering that the modelling languages and formalisms can be integrated on the syntax and semantics level, and that analysis techniques or the results of the analysis algorithms can be combined, we conceive three general composition approaches:

1. *Model composition (white-box composition)* is the analysis input-model composition realised by language integration (i.e., the definition of a new language from a set of individual languages, for example, by metamodel unification or weaving) [Cla+14; GRS09]. The internals of the composed individual models are exposed at an arbitrary level of detail and open for modifications and for analysis. Note that language composition is not always necessary for model composition. If there is a joint language for both models, or a transformation to a joint language, there is no need for language composition. For example, see "composition by transformation into a joint formalism" in Chapter 11 of this book [Hei+21].

2. *Result composition (black-box composition)* is the composition of the analysis results. The internals of the models remain encapsulated, only explicitly defined interoperability interfaces are used to access the target analysis and render back the results. Usually, a user-facing model is translated or mapped (e.g., by a model transformation realising *semantic translation* [HR04]) into a concrete model of the target analysis formalism, and then the results of the analysis are lifted back to the level of the user-facing model. Various types of black-box composition are possible, ranging from single analysis orchestration over combined analysis orchestration to sequential analysis orchestration of black-box analysis tools by exchanging results. (cf. Chapter 5 of this book [Hei+21]).

3. *Analysis composition (grey-box composition)* is the composition of the analysis techniques by orchestrating the steps of two or more analysis algorithms. Internal knowledge of models may be partially exposed through interfaces to guide the coordination, but the composition remains modularised. For example, see "composition by co-simulation" in Chapter 11 of this book [Hei+21].

In the white-box approach, the integration of two or more languages may require additional information in the form of a correspondence between the syntax and/or semantics of the constituent languages. It accommodates highly customised composition semantics, but it is not easily extensible, and it is easily applicable only if we have a high overlap between languages. The UML is a well known exemplar of a compound language resulting by the integration of several modelling formalisms properly revisited.

There has also been significant work in the language semantics community on creating modular language definitions that can be combined to form new languages. This includes work related to algebraic specification [Bra+01], rewriting logic semantics [MB04; BM05], modular structural operational semantics (MSOS) [Mos99; Mos02], implicitly-modular structural operational semantics (I-MSOS) [MN08], monads in denotational semantics [Mog89; Mog91; Esp95], abstract state machines [KP97], and the K framework [HŞR07; RS10]. This work has tended to focus on methods for defining reusable language feature modules (e.g., the ability to elide unused parts of the configuration in MSOS, the use of context completion in K), which can then be reused in the construction of a new or extended language. These would also qualify as white-box approaches since they work directly over the formal definitions of the languages.

Black-box composition keeps the composition highly modular, allowing arbitrary analysis tasks to be carried out and the results lifted back to the user-facing domain level as long as they conform to the interfaces. A typical example of this approach is the common practice of translating a user-facing model (including some temporal logic properties) into a model checker input, and then translating back the counterexample into concepts of the user-facing model language. However, because black-box composition cannot rely on internal structure of models,

it can only support a fixed composition semantics that is dictated by the corresponding semantic mapping(s) and that might be too restrictive.

The grey-box approach represents a whole spectrum of grey shades in between the white-box and black-box approaches realised via model-based analysis coordination [Cla+14]. Coordination can be achieved implicitly (*implicit coordination*), via sharing concepts with the same semantics; the corresponding models do not exchange information explicitly, but reason about artefacts related to shared semantic concepts. Coordination can also be achieved via sharing of concepts with different semantics; in this case, the corresponding models have to exchange information explicitly via interfaces (*explicit coordination*). The information exchanged can be data or control based, and requires an orchestration model (and therefore an orchestration formalism). A typical example of coordinated analysis is co-simulation where the coupled and possibly interacting simulations of two or more models up to a fixed point can create more detailed results. Hence, grey-box composition takes the best of the first two approaches and works well for highly heterogeneous languages, but requires sophisticated technicality of language orchestration engines.

In order to combine together multiple analysis tools and, therefore, combine multiple results, these forms of composition can be concretely realised by adopting specific orchestration strategies of the analysis tools involved (see Chapter 5 of this book [Hei+21] for more details). Chapter 11 of this book [Hei+21] illustrates how to implement the different forms of composition by discussing examples of concrete composition operators.

4.5 A Mathematical Characterisation of Models, Analyses and Composition

Many of the concepts we have described in the previous subsections are rather well known, and have all been dealt with in the practical realisation of modelling processes and engineering tools. However, to our knowledge, a general and unifying view on how to deal with composition of analysis, and how composition of analysis relates to compositionality of models and their semantics, contexts, and analysis algorithms, does not exist yet.

In order to provide a precise understanding of how to put all these elements in relation, this section provides a reference conceptual framework for the classes of composition we have identified so far. For that purpose, we use mathematical constructs that allow us to precisely define the effects, but of course need to be embellished in very individual forms in the various domains of software systems, analysis techniques, etc. Here we only give very short examples.

4.5.1 Model

Section 2.1.2 of this book [Hei+21] describes the concept of models where modelling formalisms provide a syntax, here called *Syn*, and a semantic domain, here called *Sem*, that provides meaning for syntactic elements. We formalise meaning as a semantic mapping [HR04]:

$$M : Syn \rightarrow Sem.$$

In a mathematical setting, the semantic domain describes an infinite set of possible realisations. For simplicity, one might think of all possible “implementations”. As the semantic domain is infinite, usually that semantic mapping is just a mathematical construction and has no algorithmic executability. Semantics in that sense serves as background for a precise definition of the desired properties that can then be proven either precisely or approximately through appropriate algorithmic analysis techniques.

Modelling languages are usually designed to describe or constrain the set of possible implementations. Therefore, by definition, a modelling language differs in its purpose from a programming language, where usually a deterministic execution is desired. The mathematical semantics of the

model should therefore reflect that it is a constraint on the set of implementations. To capture this, the semantics definition is refined to a set based approach:

$$M : Syn \rightarrow \mathcal{P}(Sem)$$

One model therefore describes a set of possible implementations. For example, a nondeterministic automaton describes a set of accepted words, a class diagram describes a set of valid object structures, and usually a behavioural description, such as an activity diagram or a Petri net describes a set of traces. If the mapping M is appropriately defined, then mathematically a number of constructs can be easily defined. For example, a model $m \in Syn$ is *consistent*, exactly if $M(m) \neq \emptyset$. Or a model m_2 is a *refinement* of another model m_1, if $M(m_2) \subset M(m_1)$.

As a consequence, it is also relatively easy to define the semantics of two (and thus arbitrary many) models m_1, m_2, that describe different aspects of a system simply by using the set of implementations that obey both models (i.e., intersection): $M(m_1 + m_2) = M(m_1) \cap M(m_2)$. This property allows us to, in the following discussion, only look at the single model, instead of the usually existing set of individual artefacts developed during the project.

This general principle of semantic definitions can be applied to each kind of syntactic artefact that is used during the development process, even if the artefacts are described in different languages. This in particular includes *property* definitions and *context models* as well as models of the system itself. For simplicity, we assume that for each language Syn_i at hand, we have an appropriate mapping M_i:

$$M_i : Syn_i \rightarrow \mathcal{P}(Sem)$$

This also serves as a nice mathematical integration of different modelling languages on a semantic level. Please note that if, e.g. in an industrial setting, various different models of different modelling languages are used, an integrated semantic domain is not easy to construct. In [Bro+09a; Bro+09b], such an effort was made for object-oriented systems as a basis for UML models.

4.5.2 Analysis

Mathematically, an analysis technique A has the very same signature as a semantic mapping M. It analyses a model from the modelling language Syn and produces a result R of an appropriate *result domain Res*:

$$A : Syn \rightarrow Res$$

As we discussed already, the purpose of an analysis, however, differs from the semantic mapping M: usually the result domain Res is a rather simple domain, covering a huge abstraction of what the original model described. Typical semantic domains for Res are:

- Boolean, which means that the analysis checks whether a property is true or false,
- Real Numbers $\mathcal{R}$, which means that the analysis measures some kind of fitness,
- A visual representation of boolean or real numbers, which means that the analysis is mainly dedicated for exhibiting certain information to the user.

Of course, more forms of analysis techniques are possible, especially if one analysis technique produces only a subresult used in another analysis technique.

Because Res usually consists of finite, computable objects, we are interested in algorithmically executable analysis techniques A as well. In complex situations, this interest in algorithmic execution often prevents to directly use the semantics domain Sem. In that sense, we might see analysis techniques to be algorithmic executable abstractions of the semantics, and it then makes sense to have several analysis techniques for different purposes available.

We furthermore might be interested in extending an analysis algorithm by an explicit definition of the desired *properties* (in language Syn_2):

$$A : Syn \times Syn_2 \rightarrow Res$$

Chapter 9 of this book [Hei+21] provides examples of this form of analysis.

The correctness of an analysis technique can be reasoned about. For example: a model $m \in Syn$ fulfils a binary property definition $p \in PL$ of a property language PL exactly if $M(m) \subseteq M_{PL}(p)$. An analysis technique A is sufficient if, for all models $m \in Syn$ and for all properties $p \in PL$, it holds that $A(m,p) \Rightarrow (M(m) \subseteq M_{PL}(p))$. Please note that this definition only demands an implication, because it may be that the property holds, but the analysis technique may fail to verify this. Based on these considerations we may even compare the quality of analysis techniques according to their results. Assuming that both A_1 and A_2 are correct as defined above: A_2 is better than A_1, if it is more accurate, i.e. $\forall m, p : A_1(m,p) \Rightarrow A_2(m,p)$.

4.5.3 Composition

Composition has many different facets. Therefore, we need to be clear on what is to be composed: Components in the system, models about the system, languages that describe different viewpoints on the system, and finally analyses that calculate parts of the results about models.

In this chapter, we concentrate on the composition of analyses and therefore at first ignore that typically the system itself is also composed. In the following, we simply assume that all models and property definitions describe the same component. This simplification avoids the necessity to compose semantic domains as well as semantic mappings. As a remark: Otherwise we would need a composition technique on the semantics of the domain as well, which is of course possible but complicates the following considerations unnecessarily. We simply assume that all semantic mappings directly go to the same semantic domain Sem.

We also keep the above described simplification, that we look only at one model, because we assume that we know how to semantically compose models. The discussion below includes all forms of models, i.e., models describing the system, models describing the context of the system and potentially also models describing interaction between both.

In the following, we give examples and mathematically define the notion of composition of analyses according to the three forms of composition informally introduced in Section 4.4.

Simple result composition

Given two analysis techniques A_i, $i \in \{1,2\}$ producing individual results in their own domains Res_i based on the same model m, we can define a *result composition* if an appropriate operator $\odot$ is available:

$$A : Syn \rightarrow Res_1 \odot Res_2$$

by

$$A(m) = A_1(m) \odot A_2(m).$$

As each analysis is conducted in isolation in a black-box manner and only the results are composed this adheres to the form black-box composition.

Model decomposition and result composition

We decompose a model $m = m_1 \odot m_2$ and then can define

$$A(m_1 \odot m_2) = A_1(m_1) \odot A_2(m_2).$$

This black-box composition together with the decomposition of models is very powerful, but potentially difficult to achieve in practice. It may be that in practice, a mixture may apply:

instead of decomposing a model into disjoint elements, it may be helpful to use algorithmically executable abstraction functions $\alpha_i : Syn \to Syn$, e.g. slicers, forget functions etc., and apply the following composition:

$$A(m) = A_1(\alpha_1(m)) \odot A_2(\alpha_2(m)).$$

This however works best if all available information is used, which means that no information should be lost under the two abstractions, i.e., $M(m) = M(\alpha_1(m)) \cup M(\alpha_2(m))$.

Please note that it may of course be possible for each A_i to be parameterised with its own property definition language, then obviously different properties can be considered.

Sequential composition

Parameterisation can also be used to embed the results of one analysis technique into the computation of another analysis technique. We can speak of *sequential composition* of analysis techniques when the following applies:

$$A(m) = A_2(m, A_1(m)),$$

where the second analysis A_2 consumes the results of the first and produces the overall result. From a functional point of view, we might also argue that the analyses themselves are composed by $A = \lambda m.A_2(m, A_1(m))$. Sequential composition, however, still adheres to the form black-box composition, if only results are exchanged between black-box analyses. If there is internal knowledge exposed by orchestrating the steps of the analyses this is considered grey-box composition. It might even be that several analysis techniques depend mutually on their results.

Mutually improving analysis composition

This shows a technically very interesting dependency, that in practice happens quite often. An example is analysis coupling until a fixpoint is reached (cf. Section 5.7 of this book [Hei+21]). The formal definitions would have the form:

$$A_i : Syn \times Res_{3-i} \to Res_i \quad (i \in \{1, 2\})$$

$$A : Syn \to Res$$

$$A(m) = (r, s) \ where \ (r, s) = (A_1(m, s), A_2(m, r)).$$

This is an equational definition for the results r and s that needs a careful consideration to understand what the possible solutions are. Typically the mutual dependencies need to be handled in an iterative, potentially approximating manner.

This works particularly well when, for example, an analysis technique A_1 can already deliver initial results with an "empty" input r^0 and further iterations improve the result in a desired direction. Formally, we derive an approximation using a series of results r^n, s^n, where for each iteration step $n \in \mathcal{N}$ the next step is computed by $s^n = A_2(m, r^n)$ and $r^{n+1} = A_1(m, s^n)$ until the iteration can stop.

Again, mutually improving analysis composition adheres to the form black-box composition, if only results are exchanged between black-box analyses. If there is internal knowledge exposed by orchestrating the steps of the analyses this is considered grey-box composition.

Simulation composition

Simulation with time progress can be seen as a very special case of the above definition, where the analysis techniques are not iteratively rerun, but the results r^n, s^n are iteratively constructed in a stepwise manner.

In this grey-box composition, we probably have a timed structure on the result domain, either in a stepwise manner $Res = (\mathcal{N} \to X)$ or in a continuous manner $Res = (\mathcal{R}^+ \to X)$, where

both use a set X of messages or events or values. Furthermore, the analysis techniques must be compositional in the sense that they do not use input of a specific time point t to produce output of a time point t_2 that is earlier or equal to t, i.e., $t_2 > t$ must hold. Mathematical theories for this kind of timing behaviour are for example given in Ptolemy [Eke+03], Focus [BS01; RR11] or Abadi/Lamport's TLA [AL90].

4.5.4 Composition of Contexts

As one of the components of an analysis, a context can appear as parameter to the analysis tool, or a context model C can be composed with the system model M, for example, to turn an open system model into a closed system model $C[M]$ for behavioural analysis. In the case of a composed system model $M = M_1 \otimes M_2$, we can consider contexts C_1, C_2 for the component models M_1, M_2, or a composite context $C = C_1 \otimes_c C_2$, and form the analysis model in two ways:

$$(C_1 \otimes_c C_2)[M_1 \otimes M_2]$$

or

$$C_1[M_1] \otimes C_2[M_2].$$

A challenge for future research is to identify conditions under which to choose one form over the other.

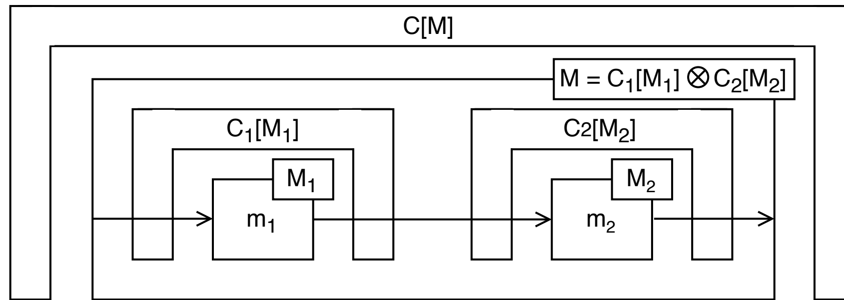

Figure 4.2: Context composition

A context may only provide part of the information needed to describe operating conditions. Thus, composition with a context can be iterated, incrementally adding contextual information. This is illustrated in Figure 4.2. Here, component models m_1 and m_2, with respective interfaces M_1 and M_2, are enclosed in contexts C_1 and C_2, respectively, forming models represented by $C_1[M_1]$ and $C_2[M_2]$ (context-model composition). Then, model $M = C_1[M_1] \otimes C_2[M_2]$ is formed by composing the resulting models (model-model composition). M may still have undetermined contextual elements. These can be provided by further composition with context C to obtain

$$C[(C_1[M_1] \otimes C_2[M_2])].$$

4.5.5 Compositionality of Property Satisfaction

A challenging question about all of these compositions is understanding conditions under which properties are preserved. A related challenge is designing an analysis technique in such a way, that no potential forms of use of a model, i.e., no forms of composition with other models, invalidate the analysis result.

We can formalise that as follows: Given an analysis result $r = A(m)$, composition with any other model m_2 should retain (or even improve) the result, e.g. in a simplified form, it holds $r = A(m \odot m_2)$.

This, however, is often not the case in practice. For example, for performance models, adding additional components usually reduces the performance of the already deployed components. To

some extent this has to do with difficulties of decomposing certain kinds of analysis techniques without pre-defining certain additional knowledge, for example, dedicated slots of computing time attached to each of the submodels.

It is also possible to consider an alternative direction, by using analysis techniques that do not only produce results, but also clarify the necessary conditions for the context of a modelled component in order to operate according to the desired properties. In this case, the analysis technique is potentially also parameterised by a property definition used as a parameter and produces as a result another property definition for the context, which then can be fed as a necessary property for the models of the context. Thus, for an existing property definition language PL we have analysis techniques of the form:

$$A : Syn \times PL \rightarrow PL$$

successively or iteratively applied to the various models as described above producing improved property definitions over time.

The nice thing with analysis techniques delivering property conditions about a modelled component is that, for example, reusable library components can be documented with this kind of usage conditions and newly defined components can be checked for compliance.

All these general considerations may work for certain kinds of properties, but certainly not for all. For example, security properties are usually not easily compositional.

4.6 Examples of Formalisms, Composition Issues, and Current Practice

To give a concrete idea of the concepts and relations discussed in the previous sections we give an overview of several formalisms and associated modelling and analysis tools. The formalisms range from general purpose modelling systems (rewriting logic, abstract state machines), formalisms designed for modelling specific aspects (hybrid automata, Palladio), and formalisms for coordination and composition (BCOoL). Rewriting logic is a general purpose formalism that supports language and model composition, and all three forms of composition introduced in Section 4.4, especially for concurrent/distributed systems. Abstract state machines is a general purpose formalism for functional behaviours, supporting black-box (result) composition. Palladio is an approach and toolset for software architecture modelling and analysis of quality properties, supporting model, result, and analysis composition. Hybrid automata is a formalism composed from discrete and continuous models of behaviour that can be considered as model composition. Grey-box (analysis) composition of hybrid systems is supported by multiple tools. GEMOC Studio is a framework for developing and composing *domain-specific modelling languages* (DSMLs). Analysis composition in GEMOC Studio is provided by coordination mechanisms specified in BCOoL.

4.6.1 Rewriting Logic and its Realisation in the Maude Language and System

Rewriting logic [Mes92; Mes12] is a logic for reasoning about change over time using rewrite rules. Maude [Cla+07; Dur+19] is a rewriting logic language and toolset providing an efficient implementation that supports executable specification and analysis of concurrent and distributed systems.[1] Similar to programming languages, Maude is a general purpose modelling language with models that can be used for simulation or answering the simple question "Does the model run?". Being based on a formal logic, many other analyses are available as well.

Rewrite theories (Maude modules) can be used for specifying many aspects. For system models the structure/architecture is represented by terms of an equational theory and the dynamics/behaviour is specified by local rewrite rules that specify how a system in a given state evolves.

[1] Maude is available at `http://maude.cs.uiuc.edu/`.

Context models can be represented using terms with "holes", by adding constraints to execution states, or by adding an explicit context component such as an environment or intruder model. Properties are specified using equationally defined boolean functions. Properties of state/system structure can be specified for static analysis, or for use as state properties in *linear temporal logic* (LTL) formulas for the model checker. Execution traces can be captured using reflection or by instrumenting execution states (and augmenting the rewrite rules to collect information). This allows properties of traces to be equationally defined, and checked by evaluation.

The metatheory of rewriting logic gives a foundation for analysis algorithms implemented in Maude. Static/structural analysis tools include the Church-Rosser checker, the coherence checker [DM12], and the termination tool [DLM08]. Maude directly supports several forms of dynamic/behaviour analysis. Prototyping/testing is supported by executing rules (modulo strategies) using the rewrite engine. The `search` command provides reachability analysis (can a state satisfying a given property be reached, and if so how). The built-in function `modelCheck` allows the user to check a system specification for satisfaction of LTL formulas where state properties are arbitrary equationally defined boolean functions. The Maude LTLR model checker [BM15] is an explicit state model checker supporting analysis of *linear temporal logic of rewriting* (LTLR) properties that involve both events (rule applications) and state predicates, including mixed properties such as fairness. The Real-Time Maude language and tool [ÖM07] supports specification and analysis of real-time and hybrid systems. Available analysis techniques include timed rewriting for simulation purposes, search, time-bounded and unbounded LTL model checking, and *timed computation tree logic* (TCTL) model checking.

Rewriting logic supports the formalisation of many forms of composition of models and of analyses, including the forms discussed in Section 4.4 and Section 4.5. The following are some examples.

1. *Composition of theories by inclusion, parameterised module instantiation, or terms in a module algebra.* Here is an example from the Soft Agent modelling framework [Tal+16]. The parameterised module `{SOLVE-SCP{Z :: VALUATION}` defines a soft constraint solver `solveSCP` using a valuation function specified in modules realising the parameter theory `VALUATION`. The module `VAL-Y-PATROL-ENERGY` imports two `VALUATION` modules `VAL-ENERGY` and `VAL-Y-PATROL` and forms a lexicographic composition of their valuation functions. A module `SCENARIO` imports a module defining a model of patrolling bots and the module `SOLVE-SCP{val2ypatrolenergy}` with the valuation parameter `Z` instantiated to `VAL-Y-PATROL-ENERGY`. `val2ypatrolenergy` is a view mapping elements of the theory `VALUATION` to their instantiation in `VAL-Y-PATROL-ENERGY`. In the `SCENARIO` module configurations to be tested and analysed are defined.

2. *Composition of models (syntax level) by term formation.* In [NT20], the operation `[app ; intruder]` is used to compose an application model, `app`, with an intruder model, `intruder`, to enable search for possible attacks. We can use this composition to illustrate the general analysis judgement,
$$M, C \vdash_T Q \leadsto A$$
of Section 2.2. Here M is the application model `app` and C is context model `intruder`. The technique T is search parameterised by the form of answer desired. The query Q is a predicate characterising attack states. The answer A can be either a boolean (yes, an attack state is reachable), a witness attack state, or an execution trace leading to an attack state. If the search space is finite, the answer could also be the number of (unique) attack states, the set of attack states, or a set of execution traces containing a trace for each reachable attack state.

3. *Algebraic and logical composition of properties.* Assume $P_1(m)$ and $P_2(m)$ are properties of models ranged over by m. Then, $P(m) = P_1(m) @ P_2(m)$ defines the composition of the results of evaluating the properties using operation @.

4. *Composition of rule rewriting with external simulators.* An example is a (co)-simulation of a cyber-physical agent behaviour where the cyber (planning) behaviour is simulated in Maude and the physical behaviour (drone or autonomous vehicle) is simulated using a special purpose flight or vehicle simulator [Mas+17]. Simulators are coordinated by meta-level rules and a message passing protocol. In this composition rewriting and simulation steps are interleaved with rewriting results passed to the simulator and simulation results passed back to the Maude. This interleaving with exchange of information can be viewed as an instance of the *Mutually improving results composition* discussed in Section 4.5.3. Recall the equation to solve is

$$A(m) = (r, s) \; where \; (r, s) = (A_1(m, s), A_2(m, r)).$$

In our example, m is the system model, s a command, r the system state, A_1 is the simulator which updates the state according to the new command, A_2 is the cyber/Maude simulate that decides the next command given the current state. So with r_0 the initial state, we have $s_0 = A_2(m, r_0)$, $r_1 = A_1(m, s_0)$, and so on. With a log in the state, this can incrementally generate a trace, or performance measures such as average or minimum distance between vehicles, (average) energy used per task, etc.

5. *Symbolic search (narrowing) composes rewriting and unification (equation solving).* Here unification is used to match rule premises with state patterns that represent potentially infinitely many specific states. The Mauda NPA protocol analysis tool [EMM06] uses this composition to determine if a given attack pattern can be realised in a system running one or more instances of given cryptographic security protocols.

6. *Rewriting modulo constraints composes rewriting with* satisfiability modulo theories *(SMT) constraint solving.* In this case, states are pairs consisting of a pattern and a constraint that finitely represent all pattern instances that satisfy the constraint. Constraints are accumulated as rewrite rules are applied. An SMT solver is invoked to check that a constrained state is consistent. An example use is to model timing properties of distance bounding and other protocols as constraints rather than concrete numbers [NTU19].

In the above, 1-2 are examples of white-box composition, 3 exemplifies black-box composition, and 4-6 are examples of grey-box composition.

4.6.2 Abstract State Machines and the ASMETA Analysis Toolset

Abstract state machines (ASMs) [BS03; BR18] are an extension of *finite state machines* (FSMs) where unstructured control states are replaced by *states* comprising arbitrary complex data (i.e., domains of objects with functions defined on them), and *transitions* are expressed by named parameterised transition rules (or simply rules) describing how the data (state function values saved into *locations*) change from one state to the next. ASM models can be read as "pseudocode over abstract data" with a *well-defined semantics*: at each computation step, all transition rules are executed in parallel, leading to simultaneous (consistent) updates of a number of locations. This basic notion of ASM has been extended to synchronous/asynchronous multi-agent ASMs for the design and analysis of distributed systems.

ASMs are primarily tailored to the formalisation and analysis of functional system behaviour via an iterative design process based on model refinement. Tools supporting the process are part of the ASMETA (ASM mETAmodeling) toolset[2] and provide different V&V activities (such as model simulation, scenario-based simulation, property verification by model checking and runtime verification to name a few). Most of these tools provide analysis support for ASMs by *semantic mapping* [GRS09; HR04], i.e., via model transformations that realise semantic mappings from

[2] ASMeta is available at `http://asmeta.sourceforge.net/` and `https://asmeta.github.io/`.

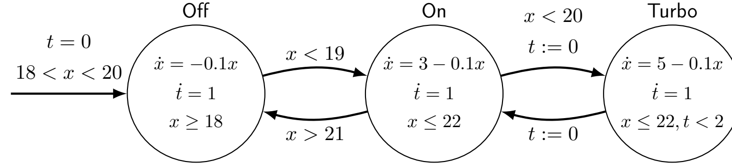

Figure 4.3: A hybrid automaton H

ASM models (edited using the textual language AsmetaL [GRS08]) to the input formalism of the target analysis tool, and then lift back the results of the analysis to the ASM level. Thus, the type of composition commonly realised in the ASMETA analysis toolset is *black-box*. More details on the specific composition strategies adopted in the ASMETA toolset are given in Chapter 5 of this book [Hei+21].

4.6.3 Palladio

Palladio is a tool-supported approach to modelling and analysing software architectures for various quality properties [Reu+16]. Details on Palladio's modelling language *Palladio Component Model* and toolset Palladio-Bench are given in Chapter 11 of this book [Hei+21]. In the context of Palladio, different forms of composition as introduced in Section 4.4 are applied. For example, IntBIIS [Hei+17] is an approach for extending Palladio architectural models by business process models to simulate the mutual performance impact of software systems and business processes. IntBIIS therefore conforms to model composition. The *Power Consumption Analyzer* (PCA) [Sti18] uses the results of Palladio's software architecture simulation (mainly utilisation of resources) to forecast power consumption of software systems at the architecture level. PCA therefore conforms to results composition. OMPCM [HMR13] integrates the OMNeT++-based network simulation framework INET with the architecture-level software performance prediction of Palladio. OMPCM therefore conforms to analysis composition.

4.6.4 Hybrid Automata

Hybrid automata [Hen00; LSV03] are finite state machines extended with continuous variables. Hybrid automata are widely used to specify cyber-physical systems that exhibit both discrete and continuous behaviour. Such systems include automotive, avionics, robotics, and medical systems. In a hybrid automaton, the discrete part of the system is specified using a finite state machine with discrete states (called *modes*) and transitions (called *jumps*), and the continuous part of the system is modelled using continuous real functions or *ordinary differential equations* (ODEs) over continuous state variables. The values of continuous variables can also be changed (or reset) when jumps happen. The parallel composition of hybrid automata is defined by synchronising jumps with common "actions" in a way similar to the case of finite state machines.

Figure 4.3 shows a hybrid automaton modelling a simple thermostat system, adapted from [Hen00]. Two (continuous) variables x and t represent the temperature and the timer, respectively, and three (discrete) modes off, on, and turbo represent the status of the heater. Initially, the mode is off, the timer t is 0, and the temperature x is any value between 18 and 20. The values of x and t change according to the ODEs for each mode, while satisfying the *invariant* conditions of the mode. For example, in the turbo mode, x and t change according to $\dot{x} = 8 - 0.1x$ and $\dot{t} = 1$ as long as the invariant conditions $x \leq 22$ and $t \leq 2$ hold. A jump between two modes can be taken if the guard condition is satisfied: e.g., a jump from on to turbo can happen whenever $x < 20$ holds, and in this case the value of t is reset to 0.

The behaviour of a hybrid automaton is given by continuous trajectories of modes and variables over time. Formally, each state of a hybrid automaton is a pair $(q, \vec{v})$ of a mode $q \in Q$ and a real-valued vector $\vec{v} \in \mathbb{R}^n$, where Q denotes a finite set of modes and $\vec{v} = (v_1, \ldots, v_n)$ denotes the

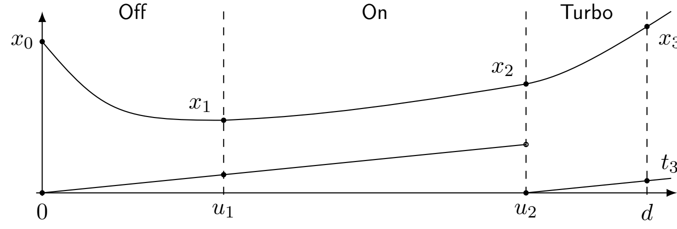

Figure 4.4: A trajectory

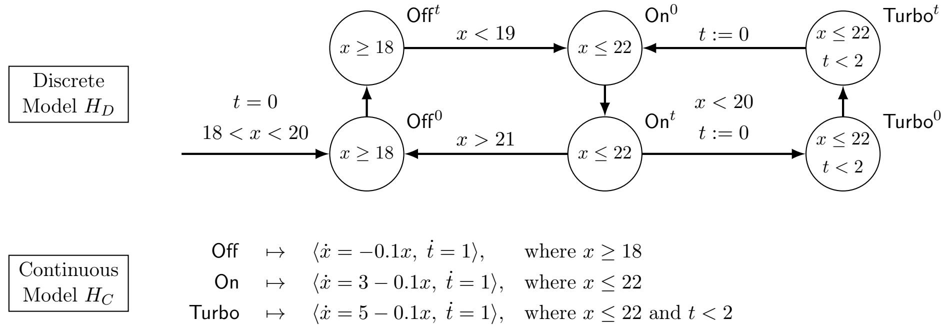

Figure 4.5: A composition $H = H_D \otimes H_C$

values of the continuous variables $x_1, \ldots, x_n$. A finite trajectory of length $d \geq 0$ is then a function $\tau : [0, d] \to Q \times \mathbb{R}^n$ that describes the continuous changes of the states in the time interval $[0, d]$. Excluding Zeno behaviour (with infinitely many jumps in a finite amount of time), a finite trajectory only involves a finite number of discrete jumps in the interval $[0, d]$. For example, a trajectory τ for the thermostat system is shown in Figure 4.4. Initially, $\tau(0) = (\mathsf{off}, (x_0, 0))$. It involves a jump from off to on at time u_1, and a jump from on to turbo at time u_2.

A hybrid automaton can be considered as the *model composition* of a finite state machine and a continuous dynamical system. Consider the thermostat hybrid automaton H above. As shown in Figure 4.5, the discrete part is the nondeterministic state machine H_D that abstracts from the continuous dynamics. Each mode m in the hybrid automaton H is separated into two states m^0 and m^t in H_D, where m^0 and m^t correspond to the beginning and the end, respectively, of a trajectory fragment with mode m. Any trajectory of H corresponds to a path in the state machine H_D. For example, the trajectory in Figure 4.4 corresponds to the path: $(\mathsf{off}^0, x_0, 0), (\mathsf{off}^t, x_1, u_1), (\mathsf{on}^0, x_1, u_1), (\mathsf{on}^t, x_2, u_2), (\mathsf{turbo}^0, x_2, 0), (\mathsf{turbo}^t, x_3, t_3)$. The continuous part is the continuous dynamical system H_C that abstracts from the transition structure. As expected, each trajectory fragment with mode m for H is a valid signal of H_C.

The safety verification problem is to check whether there exists an "error" trajectory that violates safety requirements. As usual, there are different ways to specify the safety requirements of hybrid automata, such as invariant properties of reachable states [Hen00], temporal logic properties of continuous trajectories [MN04], etc. The safety verification problem is in general undecidable for hybrid automata [Hen00]. Nevertheless, there exist several tools that can approximately verify the absence of error trajectories up to given bounds for different classes of safety properties, including SpaceEx [Fre+11], HyComp [Cim+15], Flow* [CÁS13], dReach [Kon+15], StlMC [BL19], etc. Each of those tools provides its own modelling language to specify hybrid automata. It is worth noting that these modelling languages usually have different syntaxes but have the same semantics, namely, hybrid automata.

Safety verification algorithms for hybrid automata sometimes exploit this composition relation to combine different analysis techniques for discrete and continuous dynamical systems. Consider a hybrid automaton $H = H_D \otimes H_C$. An error trajectory exists in the hybrid automata H, if a corresponding path exists both in the discrete part H_D and in the continuous part H_C. Based

on this observation, we can first find an erroneous sequence in H_D, e.g., using an SMT-based model checking algorithm for finite state machines, and then try to build a concrete continuous trajectory, e.g., using linear/non-linear real arithmetic solvers or ODE solvers. For example, SMT-based techniques for hybrid automata [Cim+15; Kon+15; BL19] can be characterised as this *analysis composition* approach, where the orchestration mechanism is the DPLL($\mathcal{T}$) SMT framework.

4.6.5 The GEMOC Studio and BCOoL

The GEMOC Studio[3] provides generic components through Eclipse technologies for the development, integration, and use of heterogeneous executable modelling languages [Bou+16]. This includes

- metaprogramming approaches and associated execution engines to design and execute the behavioural semantics of executable modelling languages,
- efficient and domain-specific execution trace management services, model animation services,
- advanced debugging facilities such as forward and backward debugging and a comprehensive timeline, and
- coordination facilities to support concurrent and coordinated execution of heterogeneous models.

In particular, the GEMOC studio comes with *Behavioral Coordination Operator Language* (BCOoL) [Lar+15], a metalanguage to explicitly specify coordination patterns between heterogeneous languages. It actually reifies coordination patterns between specific domains by using coordination operators between the DSMLs used in these domains. These patterns are captured at the language level, and then used to derive a coordination specification automatically for models conforming to the targeted DSMLs. The coordination at the language level relies on a so-called language behavioural interface (making the composition *grey-box*). This interface exposes an abstraction of the language behavioural semantics in terms of *events*. Finally, an heterogeneous execution engine, integrated to the GEMOC studio, can be configured by the coordination specification between the models in order to coordinate the execution of each of the dedicated execution engines.

BCOoL provides support for co-simulation. Using BCOoL, the know-how of an integrator is made explicit, stored and shared in libraries and amenable to analysis.

4.7 Conclusion and Outlook

In this chapter, we explored how to explicitly address the compositionality of analysis and specific forms of composition. Analysis involves models, contexts, and properties. These are all expressed in languages with their own semantics. We first gave a detailed overview of these important concepts as they are fundamental and need to be managed when composing analyses and the underlying formalisms. We have distinguished three main forms of composition: (i) model composition (white-box composition), (ii) result composition (black-box composition), and (iii) analysis composition (grey-box composition). According to such classes, we then introduced a preliminary conceptual framework that defines abstract operations for analyses composition to be implemented explicitly and managed in modelling environments. We have proceeded towards this goal both with a conceptual reasoning and practical examples of their application with real-world analysis formalisms and their supported tools.

[3] GEMOC Studio is available at `http://gemoc.org/studio`.

An open research challenge is characterisation of the compositionality of the analysis satisfaction relations and property definitions along the three forms of composition we have proposed in this chapter. Additional challenges are: to identify conditions under which to choose one form of composition over the other; and to support the specification of composition by executing relations in operative workflows that may build upon the concepts proposed in this chapter.

Bibliography

[AL90] Martin Abadi and Leslie Lamport. "Composing Specifications". In: *ACM Transactions on Programming Languages and Systems* 15.1 (1990), pp. 73–132.

[BL19] Kyungmin Bae and Jia Lee. "Bounded model checking of signal temporal logic properties using syntactic separation". In: vol. 3. 2019, 51:1–51:30. DOI: 10.1145/3290364.

[BM05] Christiano Braga and José Meseguer. "Modular Rewriting Semantics in Practice". In: *International Workshop on Rewriting Logic and Its Applications, WRLA, Proceedings*. Vol. 117. 2005, pp. 393–416. DOI: 10.1016/j.entcs.2004.06.019.

[BM15] Kyungmin Bae and José Meseguer. "Model checking linear temporal logic of rewriting formulas under localized fairness". In: *Science of Computer Programming* 99 (2015), pp. 193–234.

[Bou+16] Erwan Bousse, Thomas Degueule, Didier Vojtisek, Tanja Mayerhofer, Julien DeAntoni, and Benoît Combemale. "Execution framework of the GEMOC studio (tool demo)". In: *International Conference on Software Language Engineering, SLE, Proceedings*. 2016, pp. 84–89.

[BR18] Egon Börger and Alexander Raschke. *Modeling Companion for Software Practitioners*. Springer, 2018. DOI: https://doi.org/10.1007/978-3-662-56641-1.

[Bra+01] Mark van den Brand, Arie van Deursen, Jan Heering, H. A. de Jong, Merijn de Jonge, Tobias Kuipers, Paul Klint, Leon Moonen, Pieter A. Olivier, Jeroen Scheerder, Jurgen J. Vinju, Eelco Visser, and Joost Visser. "The ASF+SDF Meta-environment: A Component-Based Language Development Environment". In: *Proceedings of CC'01*. Vol. 2027. 2001, pp. 365–370.

[BRJ98] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[Bro+09a] Manfred Broy, María Victoria Cengarle, Hans Grönniger, and Bernhard Rumpe. "Considerations and Rationale for a UML System Model". In: *UML 2 Semantics and Applications*. Nov. 2009, pp. 43–61.

[Bro+09b] Manfred Broy, María Victoria Cengarle, Hans Grönniger, and Bernhard Rumpe. "Definition of the UML System Model". In: *UML 2 Semantics and Applications*. 2009, pp. 63–93.

[BS01] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems. Focus on Streams, Interfaces and Refinement*. Springer, 2001.

[BS03] Egon Börger and Robert Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, 2003.

[CÁS13] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. "Flow*: An Analyzer for Non-linear Hybrid Systems". In: *Computer Aided Verification - 25th International Conference, CAV, Proceedings*. Vol. 8044. 2013, pp. 258–263. DOI: 10.1007/978-3-642-39799-8_18.

[Cim+15] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. “HyComp: An SMT-Based Model Checker for Hybrid Systems”. In: *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS, Proceedings*. Vol. 9035. 2015, pp. 52–67. DOI: 10.1007/978-3-662-46681-0_4.

[Cla+07] Manuel Clavel, Francisco Durán, Steven Eker, José Meseguer, Patrick Lincoln, Narciso Martí-Oliet, and Carolyn Talcott. *All About Maude – A High-Performance Logical Framework*. Vol. 4350. Springer, 2007.

[Cla+14] Tony Clark, Mark van den Brand, Benoît Combemale, and Bernhard Rumpe. “Conceptual Model of the Globalization for Domain-Specific Languages”. In: *Globalizing Domain-Specific Languages - International Dagstuhl Seminar, Revised Papers*. Vol. 9400. 2014, pp. 7–20. DOI: 10.1007/978-3-319-26172-0_2.

[Com+16] Benoit Combemale, Robert France, Jean-Marc Jézéquel, Bernhard Rumpe, Jim R.H. Steel, and Didier Vojtisek. *Engineering Modeling Languages*. Chapman and Hall/CRC, 2016, p. 398. URL: http://mdebook.irisa.fr/.

[DLM08] Francisco Durán, Salvador Lucas, and José Meseguer. “MTT: The Maude Termination Tool (System Description)”. In: *Automated Reasoning, 4th International Joint Conference*. Vol. 5195. 2008, pp. 313–319.

[DM12] Francisco Durán and José Meseguer. “On the Church-Rosser and coherence properties of conditional order-sorted rewrite theories”. In: *Journal of Logic and Algebraic Programming* 81.7–8 (2012), pp. 816–850.

[Dur+19] Francisco Durán, Steven Eker, Santiago Escobar, Narciso Martí-Oliet, José Meseguer, Rubén Rubio, and Carolyn L. Talcott. “Programming and Symbolic Computation in Maude”. In: *Journal of Logical and Algebraic Methods in Programming* (2019).

[Eke+03] Johan Eker, Jorn W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, S. Neuendorffer, Sonia Sachs, and Yuhong Xiong. “Taming heterogeneity - the Ptolemy approach”. In: *Proceedings of the IEEE* 91.1 (2003), pp. 127–144. DOI: 10.1109/JPROC.2002.805829.

[EMM06] Santiago Escobar, Cathy Meadows, and José Meseguer. “A Rewriting-Based Inference System for the NRL Protocol Analyzer and its Meta-Logical Properties”. In: *Theoretical Computer Science* 367.1–2 (2006), pp. 162–202.

[Esp95] David A. Espinosa. “Semantic Lego”. PhD thesis. 1995.

[Fre+11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. “SpaceEx: Scalable Verification of Hybrid Systems”. In: *Computer Aided Verification - 23rd International Conference, CAV, Proceedings*. Vol. 6806. 2011, pp. 379–395. DOI: 10.1007/978-3-642-22110-1_30.

[Gro06] Object Management Group. *MOF Specification Version 2.0 (2006-01-01)*. http://www.omg.org/doc 05-04.pdf. Jan. 2006.

[GRS08] Angelo Gargantini, Elvinia Riccobene, and Patrizia Scandurra. “A Metamodel-based Language and a Simulation Engine for Abstract State Machines”. In: *J. UCS* 14.12 (2008), pp. 1949–1983. DOI: 10.3217/jucs-014-12-1949.

[GRS09] Angelo Gargantini, Elvinia Riccobene, and Patrizia Scandurra. “A semantic framework for metamodel-based languages”. In: *Autom. Softw. Eng.* 16.3-4 (2009), pp. 415–454. DOI: 10.1007/s10515-009-0053-0.

[Hei+17] Robert Heinrich, Philipp Merkle, Jörg Henss, and Barbara Paech. “Integrating business process simulation and information system simulation for performance prediction”. In: *Software & Systems Modeling* 16.1 (2017), pp. 257–277. DOI: 10.1007/s10270-015-0457-1.

[Hei+21] Robert Heinrich, Francisco Durán, Carolyn L. Talcott, and Steffen Zschaler (eds.) *Composing Model-Based Analysis Tools*. Springer, 2021.

[Hen00] Thomas A. Henzinger. "The theory of hybrid automata". In: *Verification of digital and hybrid systems*. 2000, pp. 265–292.

[HMR13] Jörg Henss, Philipp Merkle, and Ralf H. Reussner. "The OMPCM Simulator for Model-Based Software Performance Prediction: Poster Abstract". In: *6th International ICST Conference on Simulation Tools and Techniques, Proceedings*. 2013, pp. 354–357.

[HR04] David Harel and Bernhard Rumpe. "Meaningful modeling: what's the semantics of "semantics"?" In: *Computer* 37.10 (2004), pp. 64–72.

[HŞR07] Mark Hills, Traian Florin Şerbănuţă, and Grigore Roşu. "A Rewrite Framework for Language Definitions and for Generation of Efficient Interpreters". In: *6th International Workshop on Rewriting Logic and its Applications, WRLA, Proceedings*. Vol. 176. 2007, pp. 215–231. DOI: 10.1016/j.entcs.2007.06.017.

[Kon+15] Soonho Kong, Sicun Gao, Wei Chen, and Edmund M. Clarke. "dReach: δ-Reachability Analysis for Hybrid Systems". In: *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS, Proceedings*. Vol. 9035. 2015, pp. 200–205. DOI: 10.1007/978-3-662-46681-0_15.

[KP97] Philipp W. Kutter and Alfonso Pierantonio. "Montages Specifications of Realistic Programming Languages". In: *J. UCS* 3.5 (1997), pp. 416–442.

[Lar+15] Matias Ezequiel Vara Larsen, Julien DeAntoni, Benoît Combemale, and Frédéric Mallet. "A Behavioral Coordination Operator Language (BCOoL)". In: *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS, Proceedings*. 2015, pp. 186–195. DOI: 10.1109/MODELS.2015.7338249.

[LSV03] Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. "Hybrid I/O automata". In: *Inf. Comput.* 185.1 (2003), pp. 105–157. DOI: 10.1016/S0890-5401(03)00067-1.

[Mas+17] Ian A. Mason, Vivek Nigam, Carolyn Talcott, and Alisson Brito. "A Framework for Analyzing Adaptive Autonomous Aerial Vehicles". In: *1st Workshop on Formal Co-Simulation of Cyber-Physical Systems*. 2017.

[MB04] José Meseguer and Christiano Braga. "Modular Rewriting Semantics of Programming Languages". In: *10th International Conference on Algebraic Methodology and Software Technology, AMAST, Proceedings*. Vol. 3116. 2004, pp. 364–378.

[Mes12] José Meseguer. "Twenty years of rewriting logic". In: *J. Algebraic and Logic Programming* 81 (2012), pp. 721–781.

[Mes92] José Meseguer. "Conditional Rewriting Logic as a Unified Model of Concurrency". In: *Theoretical Computer Science* 96.1 (1992), pp. 73–155.

[MN04] Oded Maler and Dejan Nickovic. "Monitoring Temporal Properties of Continuous Signals". In: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS, and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT, Proceedings*. Vol. 3253. 2004, pp. 152–166. DOI: 10.1007/978-3-540-30206-3_12.

[MN08] Peter D. Mosses and Mark J. New. "Implicit Propagation in Structural Operational Semantics". In: *Proceedings of SOS'08*. Vol. 229.4. 2008, pp. 49–66.

[Mog89] Eugenio Moggi. *An Abstract View of Programming Languages*. Tech. rep. ECS-LFCS-90-113. Edinburgh University, Department of Computer Science, June 1989.

[Mog91] Eugenio Moggi. "Notions of Computation and Monads". In: *Information and Computation* 93.1 (1991), pp. 55–92.

[Mos02] Peter D. Mosses. "Pragmatics of Modular SOS". In: *Proceedings of AMAST'02*. Vol. 2422. 2002, pp. 21–40.

[Mos99] Peter D. Mosses. "Foundations of Modular SOS". In: *Proceedings of MFCS'99*. Vol. 1672. 1999, pp. 70–80.

[NT20] Vivek Nigam and Carolyn Talcott. "Automated Construction of Security Integrity Wrappers for Industry 4.0 Applications". In: *The 13th International Workshop on Rewriting Logic and its Applications*. 2020.

[NTU19] Vivek Nigam, Carolyn Talcott, and Abraão Aires Urquiza. "Symbolic Timed Trace Equivalence". In: *CathyFest2019*. 2019.

[ÖM07] Peter Csaba Ölveczky and José Meseguer. "Semantics and Pragmatics of Real-Time Maude". In: *Higher-Order and Symbolic Computation* 20.1-2 (2007), pp. 161–196.

[Reu+16] Ralf H. Reussner, Steffen Becker, Jens Happe, Robert Heinrich, Anne Koziolek, Heiko Koziolek, Max Kramer, and Klaus Krogmann. *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press, 2016.

[RR11] Jan Oliver Ringert and Bernhard Rumpe. "A Little Synopsis on Streams, Stream Processing Functions, and State-Based Stream Processing". In: *International Journal of Software and Informatics* (2011).

[RS10] Grigore Rosu and Traian-Florin Serbanuta. "An overview of the K semantic framework". In: *J. Log. Algebraic Methods Program.* 79.6 (2010), pp. 397–434. DOI: 10.1016/j.jlap.2010.03.012.

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer, 1973.

[Sti18] Christian Stier. "Adaptation-Aware Architecture Modeling and Analysis of Energy Efficiency for Software Systems". PhD thesis. Karlsruher Institut fur Technologie (KIT), 2018. DOI: 10.5445/IR/1000083402.

[Tal+16] Carolyn Talcott, Vivek Nigam, Farhad Arbab, and Tobia Kappe. "Formal Specification and Analysis of Robust Adaptive Distributed Cyber-Physical Systems". In: *SFM 2016: Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems*. Vol. 9700. 2016, pp. 1–35.